1.结构体的定义与使用

结构体是一种构造数据类型,把不同类型的数据组合成一个整体
结构体的定义形式:

```
struct 结构体名
{
    结构体所包含的变量或数组
};
```

结构体是一种集合，它里面包含了多个变量或数组，它们的类型可以相同，也可以不同，每个这样的变量或数组都称为结构体的成员（Member）。请看下面的一个例子：

```
struct Student
{
    char name[20];  //姓名
    int num;  //学号
    int age;  //年龄
    char group;  //所在学习小组
    float score;  //成绩
};
```

注意：大括号后面的分号;不能少，这是一条完整的语句。

Student 为结构体名，它包含了 5 个成员，分别是 name、num、age、group、score。结构体成员的定义方式与变量和数组的定义方式相同,只是不能初始化。

结构体也是一种数据类型,它由程序员自己定义,可以包含多个其他类型的数据。像 int、float、char 等是由 C 语言本身提供的数据类型，不能再进行分拆，我们称之为基本数据类型；而结构体可以包含多个基本类型的数据，也可以包含其他的结构体，我们将它称为复杂数据类型或构造数据类型。

先定义结构体类型，再定义结构体变量

```
struct  Student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
};
struct Student stu1,stu2;
```

定义结构体类型的同时定义结构体变量

```
struct  Data
{
    int day ;
    int month;
    int year
}time1,time2;
```

直接定义结构体变量

```
struct
{
    char name[20];
    char sex;
    int num;
} person1,person2;
```

2.结构体变量的初始化

和其它类型变量一样，对结构体变量可以在定义时指定初始值。

```
#include <stdio.h>
struct  Books
{
    char  title[50];
    char  author[50];
    char  subject[100];
    int   book_id;
} book = {"C 语言", "RUNOOB", "编程语言", 123456};

int main()
{

printf("title : %s\nauthor: %s\nsubject: %s\nbook_id: %d\n",
book.title, book.author,    book.subject, book.book_id);
}
```

输出结果：

```
title : C 语言
author: RUNOOB
subject: 编程语言
book_id: 123456
```

3.访问结构成员

为了访问结构的成员，我们使用成员访问运算符（.）。

引用形式：<结构体类型变量名>.<成员名>

注意：结构体变量不能整体引用，只能引用变量成员
(一般我们是将结构体放在主函数外面 ,这里做个反例子)

成员访问运算符是结构变量名称和我们要访问的结构成员之间的一个句号。可以使用 struct 关键字来定义结构类型的变量。下面的实例演示了结构的用法：

```
#include <stdio.h>

struct
{
    char *name;  //姓名
    int num;  //学号
    int age;  //年龄
    char group;  //所在小组
    float score;  //成绩
} stu1;
int main()
{
    //给结构体成员赋值
    stu1.name = "Tom";
    stu1.num = 12;
    stu1.age = 18;
    stu1.group = 'A';
    stu1.score = 136.5;
    //读取结构体成员的值
    printf("%s 的学号是%d，年龄是%d，在%c 组，今年的成绩是%.1f!
\n",stu1.name,
stu1.num,
stu1.age,
stu1.group,
stu1.score);
    return 0;
}
```

运行结果：

```
Tom的学号是12，年龄是18，在A组，今年的成绩是136.5!
```

除了可以对成员进行逐一赋值，也可以在定义时整体赋值，例如：

```
struct
{
    char *name;  //姓名
    int num;  //学号
    int age;  //年龄
    char group;  //所在小组
```

```
    float score;  //成绩
} stu1, stu2 = { "Tom", 12, 18, 'A', 136.5 };
```

不过整体赋值仅限于定义结构体变量的时候，在使用过程中只能对成员逐一赋值，这和数组的赋值非常类似。

需要注意的是，结构体是一种自定义的数据类型，是创建变量的模板，不占用内存空间；结构体变量才包含了实实在在的数据，需要内存空间来存储。

4.结构作为函数参数
可以把结构作为函数参数，传参方式与其他类型的变量或指针类似。例如：

```
#include <stdio.h>
#include<string.h>

struct Books
{
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
};

void printBook(struct Books book)
{
    printf("Book title:%s\n", book.title);
    printf("Book author:%s\n", book.author);
    printf("Book subject:%s\n", book.subject);
    printf("Book book_id:%d\n", book.book_id);
}
int main()
{
    struct Books Book1;//声明Book1,类型为Books
    struct Books Book2;

    /* Book1 详述 */
    strcpy(Book1.title, "C Programming");
    strcpy(Book1.author, "Nuha Ali");
    strcpy(Book1.subject, "C Programming Tutorial");
    Book1.book_id = 6495407;
    printBook(Book1);
    return 0;
}
```

运行结果：

```
Book title:C Programming
Book author:Nuha Ali
Book subject:C Programming Tutorial
Book book_id:6495407
```

5.结构体数组

所谓结构体数组，是指数组中的每个元素都是一个结构体。在实际应用中，结构体数组常被用来表示一个拥有相同数据结构的群体，比如一个班的学生、一个车间的职工等。

定义结构体数组和定义结构体变量的方式类似，请看下面的例子：

```
struct Student
{
    char *name;  //姓名
    int num;  //学号
    int age;  //年龄
    char group;  //所在小组
    float score;  //成绩
}class[5];


//表示一个班有 5 个人
```

结构体数组在定义的同时也可以初始化，例如：

```
struct Student
{
    char *name;  //姓名
    int num;  //学号
    int age;  //年龄
    char group;  //所在小组
    float score;  //成绩
}class[5] =
        {
                {"Li ping", 5, 18, 'C', 145.0},
                {"Zhang ping", 4, 19, 'A', 130.5},
                {"He fang", 1, 18, 'A', 148.5},
                {"Cheng ling", 2, 17, 'F', 139.0},
                {"Wang ming", 3, 17, 'B', 144.5}
        };
```

当对数组中全部元素赋值时，也可以不给出数组长度，如：

```
struct Student
{
    char *name;  //姓名
```

```
    int num;  //学号
    int age;  //年龄
    char group;  //所在小组
    float score;  //成绩
}class[] = {
        {"Li ping", 5, 18, 'C', 145.0},
        {"Zhang ping", 4, 19, 'A', 130.5},
        {"He fang", 1, 18, 'A', 148.5},
        {"Cheng ling", 2, 17, 'F', 139.0},
        {"Wang ming", 3, 17, 'B', 144.5}
};
```

结构体数组的使用也很简单。例如，计算全班学生的总成绩、平均成绩和 140 分一下的人数：

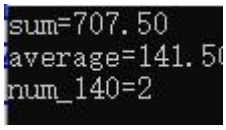
```
#include <stdio.h>

struct Student
{
    char name[20];  //姓名
    int num;  //学号
    int age;  //年龄
    char group;  //所在小组
    float score;  //成绩
}  classes[5] = {
        {"Li ping", 5, 18, 'C', 145.0},
        {"Zhang ping", 4, 19, 'A', 130.5},
        {"He fang", 1, 18, 'A', 148.5},
        {"Cheng ling", 2, 17, 'F', 139.0},
        {"Wang ming", 3, 17, 'B', 144.5}
};

int main()
{
    int i, num_140 = 0;
    float sum = 0;
    for(i=0; i<5; i++)
    {
        sum += classes[i].score;
        if(classes[i].score < 140) num_140++;
    }
    printf(" sum=%.2f\n average=%.2f\n num_140=%d\n", sum,
```

```
sum/5, num_140);
    return 0;
}
```

运行结果：



指针的简单运用

1. 了解指针

指针，是 C 语言中的一个重要概念及其特点，也是掌握 C 语言比较困难的部分。指针也就是内存地址，指针变量是用来存放内存地址的变量，不同类型的指针变量所占用的存储单元长度是相同的，而存放数据的变量因数据的类型不同，所占用的存储空间长度也不同。有了指针以后，不仅可以对数据本身，也可以对存储数据的变量地址进行操作。

2. 指针与指针变量的区别

指针：
指针就是地址，地址就是指针。
指针变量：
指针变量是变量。定义一个指针变量，是在内存中开辟一个空间，该空间里面存放地址。

如何使用：
指针更多强调的是内容（对应右值），指针变量更多强调的是空间（对应左值）。
判断一个指针和一个指针变量要通过判断它是左值还是右值
左值与右值：
  空间 对应 左值
例：（对 a 来说）

```
a = 20; //当把值赋给 a 时，是给 a 的空间写入 20。
```

  内容 对应 右值
例：（对 a 来说）

```
b = a; //把 a 的内容赋给 b。
```

3. 指针与指针类型

（1）指针的类型

从语法的角度看，你只要把指针声明语句里的指针名字去掉，剩下的部分就是这个指针的类型。这是指针本身所具有的类型。让我们看看例一中各个指针的类型：

```
(1)int*ptr;//指针的类型是 int*
(2)char*ptr;//指针的类型是 char*
(3)int**ptr;//指针的类型是 int**
(4)int(*ptr)[3];//指针的类型是 int(*)[3]
(5)int*(*ptr)[4];//指针的类型是 int*(*)[4]
```

（2）指针所指向的类型

当你通过指针来访问指针所指向的内存区时，指针所指向的类型决定了编译器将把那片内存区里的内容当做什么来看待。

从语法上看，你只须把指针声明语句中的指针名字和名字左边的指针声明符*去掉，剩下的就是指针所指向的类型。例如：

```
(1)int*ptr; //指针所指向的类型是 int
(2)char*ptr; //指针所指向的的类型是 char
(3)int**ptr; //指针所指向的的类型是 int*
(4)int(*ptr)[3]; //指针所指向的的类型是 int()[3]
(5)int*(*ptr)[4]; //指针所指向的的类型是 int*()[4]
```

（3）指针的值----或者叫指针所指向的内存区或地址

指针的值是指针本身存储的数值，这个值将被编译器当作一个地址，而不是一个一般的数值。在 32 位程序里，所有类型的指针的值都是一个 32 位整数，因为 32 位程序里内存地址全都是 32 位长。

指针所指向的内存区就是从指针的值所代表的那个内存地址开始，长度为 sizeof(指针所指向的类型)的一片内存区。以后，我们说一个指针的值是 XX，就相当于说该指针指向了以 XX 为首地址的一片内存区域；我们说一个指针指向了某块内存区域，就相当于说该指针的值是这块内存区域的首地址。指针所指向的内存区和指针所指向的类型是两个完全不同的概念。在例一中，指针所指向的类型已经有了，但由于指针还未初始化，所以它所指向的内存区是不存在的，或者说是无意义的。

（5）指针本身所占据的内存区

指针本身占了多大的内存？你只要用函数 sizeof(指针的类型)测一下就知道了。在 32 位平台里，指针本身占据了 4 个字节的长度。指针本身占据的内存这个概念在判断一个指针表达式（后面会解释）是否是左值时很有用。

4. 指针的安全问题

（1）看下面的例子：

```c
int main()
{
    char s = 'a';
    int *ptr;
    ptr = (int *) &s;
    *ptr = 1298;
}
```

指针 ptr 是一个 int *类型的指针，它指向的类型是 int。它指向的地址就是 s 的首地址。在 32 位程序中，s 占一个字节，int 类型占四个字节。最后一条语句不但改变了 s 所占的一个字节，还把和 s 相临的高地址方向的三个字节也改变了。这三个字节是干什么的？只有编译程序知道，而写程序的人是不太可能知道的。也许这三个字节里存储了非常重要的数据，也许这三个字节里正好是程序的一条代码，而由于你对指针的马虎应用，这三个字节的值被改变了！这会造成崩溃性的错误。

（2）野指针

野指针就是指针指向的位置是不可知的（随机的、不正确的、没有明确限制的）

野指针成因

A.指针未初始化

```c
#include <stdio.h>
int main()
{
    int *p;//局部变量指针未初始化，默认为随机值
    *p = 20;
    return 0;
}
```

B.指针访问越界

```c
#include <stdio.h>
int main()
{
    int arr[10] = {0};
    int *p = arr;
    int i = 0;
    for(i=0; i<=11; i++)
    {
        //当指针指向的范围超出数组 arr 的范围时，p 就是野指针
        *(p++) = i;
    }
    return 0;
}
```

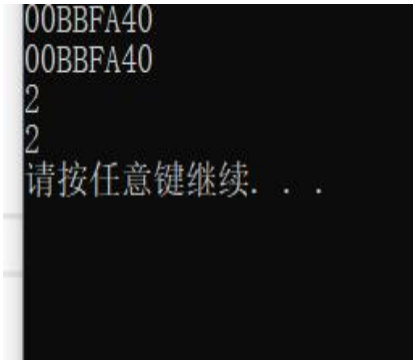
C.指针指向的空间释放

如何规避野指针的方法：
a.指针初始化
b.小心指针越界
c.指针指向空间释放即使置 NULL
d.指针使用之前检查有效性

5. 指针与数组

```c
#include<stdio.h>
int main()
{
    int arr[10] = { 1,2,3,4,5,6,7,8,9,0 };
    int* p = arr;
    printf("%p\n", arr+1);
    printf("%p\n", p+1);
    printf("%d\n",arr[1] );
    printf("%d\n", *(p + 1));
    return 0;
}
```

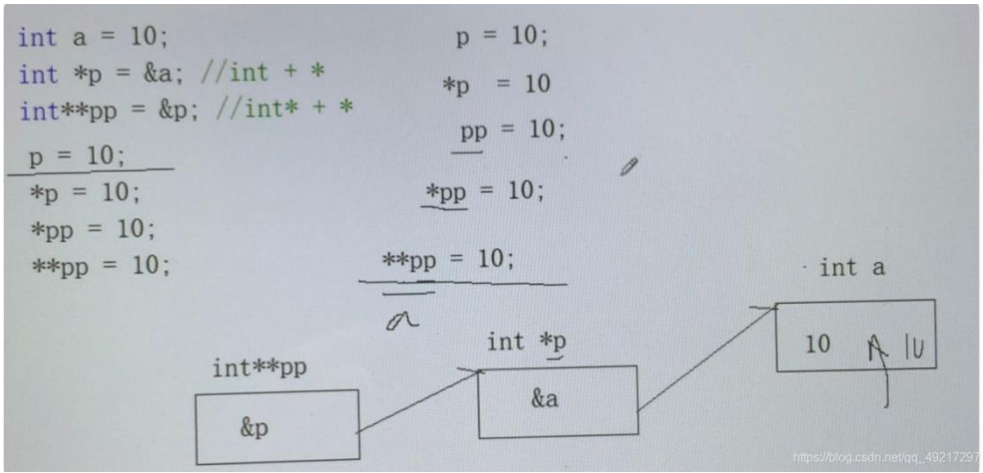
打印结果：



所以可以用指针保存数组首元素地址，使用指针进行访问数组元素。

6. 二级指针

指针变量也是变量，是变量就有地址，那指针变量的地址存放在哪里？ 就是二级指针。



对于二级指针的运算有：
*pp 通过对 pp 中的地址进行解引用，这样找到的是 p ， *pp 其实访问的就是 p .
```
int a = 20;
*pp = &a;//等价于 p = &a;
**pp 先通过 *pp 找到 p ,然后对 p 进行解引用操作：*p ，那找到的是 a .
**pp = 10;
//等价于*p = 10;
//等价于 a = 10;
```
结论：
二级指针及以上的指针自加 1 是加上 4 字节（32 位系统）或 8 字节（64 位系统）
一级指针自加 1 就是加上所指向类型的大小

7. 指针数组和数组指针

数组指针：是指针，指向数组。例：int (*arr)[10]
指针数组：是数组，数组内容存放的是指针。例：int *arr[10]
然后，需要明确一个优先级顺序：()>[]>*
所以：
(*p)[n]：根据优先级，先看括号内，则 p 是一个指针，这个指针指向一个一维数组，数组长度为 n，这是“数组的指针”，即数组指针；
p[n]：根据优先级，先看[]，则 p 是一个数组，再结合，这个数组的元素是指针类型，共 n 个元素，这是“指针的数组”，即指针数组。

8. 指针和结构类型的关系

```
struct MyStruct
{
    int a;
    int b;
    int c;
};
struct MyStruct ss = {20, 30, 40};
struct MyStruct *ptr=&ss;
```

可以声明一个指向结构类型对象的指针。
```
 //声明了结构对象 ss，并把 ss 的成员初始化为 20，30 和 40。
 //声明了一个指向结构对象 ss 的指针。它的类型是
    //MyStruct *,它指向的类型是 MyStruct。
```

请问怎样通过指针 ptr 来访问 ss 的三个成员变量？
答案：

```
ptr->a; //指向运算符，或者可以这们(*ptr).a,建议使用前者
ptr->b;
ptr->c;
```